UNIDAD DIDACTICA 4

Análisis y Búsqueda de Respuestas en Datos.

En esta unidad de trabajo vamos a tratar lo referente al análisis y la búsqueda de respuestas en datos.
Comenzamos viendo una serie de conceptos generales para después ver qué niveles existen en cuanto a analítica de datos y cuáles son las principales metodologías que se suelen emplear en minería de datos.
A continuación veremos R, un lenguaje interpretado especializado en cálculos estadísticos yen análisis de datos.
Por último conoceremos el lenguaje Python, de propósito general pero con varias librerías especializadas que le permiten trabajar muy bien analizando datos.

1. Conceptos básicos.

En esta sección veremos algunos conceptos generales que nos pueden servir como base para comprender la complejidad a la que podemos llegar a enfrentarnos cuando analizamos y buscamos respuestas en datos.
Veremos aquí un esquema/resumen para que puedas tener una vista general:

- **Lógica algorítmica:** La base para comprender cómo se definen las soluciones a problemas.
- **Combinatoria y explosión combinacional:** Bases relacionadas con la matemática discreta para comprender lo grande que puede llegar a ser el espacio de soluciones de un problema debido a la combinatoria.
- **Complejidad computacional:** El estudio de la complejidad de los problemas en función de los recursos en tiempo o memoria necesarios para resolverlos con el mejor algoritmo posible.

1.1. Lógica algorítmica.

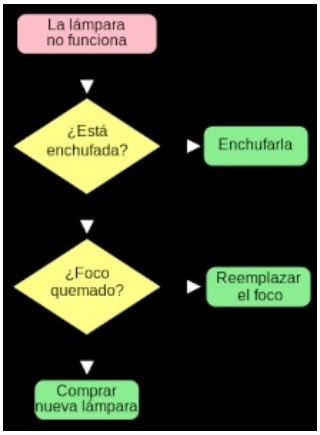
Vamos a comentar los conceptos básicos entendiendo qué es un algoritmo.
Un algoritmo es un conjunto de instrucciones que realizadas en orden permiten solucionar un problema.
Los algoritmos pueden recibir cero o más valores de entrada sobre los que trabajar, y pueden producir valores de salida o provocar algún tipo de efecto durante su ejecución.
Un algoritmo debe ser:

- **Preciso:** cada paso debe indicar qué hacer sin ambigüedades.
- **Finito:** con un número limitado de pasos.
- **Definido:** debe producir siempre el mismo resultado para los mismos valores de entrada.

Es posible definirlos con cierta independencia del lenguaje en el que finalmente se vayan a implementar, mediante diagramas de flujo o mediante pseudocódigo.

Mediante diagrama de flujo:

Podemos definir un algoritmo mediante un diagrama de flujo teniendo en cuenta que el diagrama debe tener un único nodo de inicio y al menos uno de final.



En el siguiente enlace podrás ver información que debes conocer sobre los diagramas de flujo.

Diagrama de flujo

Mediante pseudocódigo:

Podemos definir un algoritmo mediante un pseudocódigo, que se escribe en un lenguaje a medio camino del lenguaje natural y del lenguaje de programación.

Pseudocódigo de un algoritmo para sumar los enteros del 1 al 100.

establecer resultado a 0
para i := 1 hasta 100
hacer establecer resultado a resultado + i
fin

En el siguiente enlace podrás ver información que debes conocer sobre el pseudocódigo.

Pseudocódigo

En este enlace puedes ver más información sobre lo que es un algoritmo.

Algoritmo

1.2. Combinatoria.

Los algoritmos se utilizan para resolver problemas de todo tipo, para los cuales existe en ocasiones un amplísimo espacio de soluciones Tal espacio de soluciones por lo general está formado por combinaciones de posibles acciones o valores.
Por esa razón, a la hora de comprender la complejidad de los problemas a los que se enfrentan los algoritmos es fundamental adquirir un conocimiento básico sobre combinatoria, y en particular saber que es posible calcular el número de combinaciones, variaciones o permutaciones posibles en base a fórmulas bien definidas.

Permutaciones:

Podemos querer disponer N elementos en un orden determinado. Eso es una permutación.
Si queremos saber de cuántos modos distintos podemos ordenar una lista, estaremos calculando entonces el número de permutaciones.

Para el ejemplo de una lista de 3 elementos, podemos escoger cualquiera de los 3 para el primer lugar, cualquiera de los 2 restantes para el segundo lugar, y finalmente ya sólo nos queda uno para el último lugar.
Por lo tanto podemos ordenarla de 3x2x1=6 formas distintas.

Variaciones:

Podemos querer disponer N elementos de entre un total de M en un orden determinado. Eso es una variación.
Si queremos saber de cuántos modos distintos podemos tomar N elementos de un conjunto de M y ordenar esos N, estaremos calculando entonces el número de variaciones.
Por ejemplo, si en una competición participan 4 personas y queremos saber de cuántas formas posibles podrían otorgarse las medallas de oro y de plata, podemos empezar calculando los posibles puestos de los cuatro como permutaciones (4x3x2x1=24). Después nos daríamos cuenta de que, si sólo nos interesan los 2 primeros puestos, por cada posible combinación de primero y segundo nos han salido dos permutaciones (una con un tercero y un cuarto y otra en el que ese cuarto está tercero y el tercero está cuarto). De modo que si dividimos entre 2 nos sale que el número de variaciones es de 24/2 = 12.

Combinaciones:

Las combinaciones son como las variaciones (tomamos N elementos de un conjunto de M)pero sin importar en qué orden.
Si queremos saber de cuántos modos distintos podemos tomar N elementos de un conjunto de M, estaremos calculando entonces el número de combinaciones.
Por ejemplo, si queremos saber de cuántas formas podemos elegir 2 países de entre 4 posibles, podemos tomar primero el país A y después el país B, o primero el país B y después el A, siendo el mismo resultado porque como hemos dicho no importa el orden. De modo que calcularíamos el número de variaciones (12, igual que en el ejemplo anterior), y después dividiríamos entre el número de formas de ordenar los 2 países que hemos sacado(2), obteniendo un resultado final de 12/2 = 6.Como hemos dicho, existen unas fórmulas bien definidas para calcular números combinatorios (el número de variaciones, permutaciones o combinaciones según el tamaño del conjunto de partida y cuántos elementos se tomen, ya sea con o sin repetición).
Puedes encontrar más información sobre combinatoria y el cálculo de números combinatorios en el siguiente enlace:

Combinatoria

Si el concepto de combinatoria es importante a la hora de trabajar con algoritmos, más aún lo es el de **explosión combinatoria**.
Decimos que para un determinado problema aparece una explosión combinatoria cuando el número de posibles soluciones crece muy rápido a medida que aumentamos determinados valores de configuración del propio problema.
Por ejemplo, si el problema consiste en comprobar qué permutaciones de N elementos cumplen determinada condición, podemos comprobar que el tamaño del espacio de soluciones es igual al factorial de N. Por lo tanto, rápidamente nos empezaremos a encontrar tamaños extremadamente grandes incluso para valores de N relativamente pequeños:

N	N!
1	1
2	2
3	6

4	24
5	120
6	720
7	5049
8	40320
9	362880
10	3628800
100	~$9{,}33262154\times10^{157}$

Puedes encontrar más información acerca del concepto de explosión combinatoria en el siguiente enlace:

Explosión combinatoria

1.3. Complejidad computacional.

Como hemos visto, usamos algoritmos para resolver problemas, los cuales en ocasiones presentan una explosión combinacional.
Esa combinatoria produce una determinada complejidad computacional, que también es importante conocer y estudiar.
En primer lugar, hemos de tener en cuenta que cuando se estudia la complejidad computacional de un problema realmente existen dos tipos.

- **Complejidad en tiempo de ejecución:**cuánto tardaremos en resolverlo.
- **Complejidad en memoria:** cuánta memoria necesitaremos para resolverlo.

Sin embargo por lo general cuando hablamos de complejidad computacional si no se menciona nada relacionado con la memoria entonces podemos considerar que se trata de complejidad en tiempo de ejecución. Esto se debe a que son mucho más comunes los problemas cuya complejidad se dispara en tiempo de ejecución que aquellos que se disparan en memoria.
Enfocándonos ya únicamente en la complejidad computacional en tiempo, la mayor distinción que se realiza es si el problema puede resolverse en un tiempo polinómico (en función del tamaño de la entrada) o por el contrario es necesario emplear un tiempo exponencial.
También hay que tener en cuenta que los problemas se resuelven con algoritmos, para los cuales también podemos realizar un estudio sobre el tiempo y la memoria que emplean para resolver el problema para el cual se han diseñados, lo cual entra dentro del campo del análisis de algoritmos.
Por lo tanto, la complejidad de un problema viene indicada por el tiempo y/o memoria que necesite emplear el mejor algoritmo posible. En este sentido, la complejidad computacional se ocupa de estudiar la complejidad del problema como tal (no en base a algoritmos concretos).
Con ello, un problema será considerado *tratable* si se puede encontrar un algoritmo capaz de resolverlo en tiempo polinomial, e *intratable* si no existe tal algoritmo y es necesario emplear un tiempo exponencial.
Algoritmo de tiempo polinomial:
Es aquel que en el peor de los casos permite solucionar cierto problema en un tiempo determinado por un polinómio en función del tamaño de la entrada.

Por ejemplo, un algoritmo que para ordenar una lista de N elementos pueda llegar a necesitar $3N^3+N+8$ segundos en el peor caso, sería polinomial.
Evidentemente preferiríamos otro que tarde $7N^2$ segundos porque para valores grandes de Nemplea menor tiempo.
A efectos de complejidad ambos se consideran polinomiales, pero el primero sería $O(N^3)$ orden de N al cubo y el segundo $O(N^2)$ orden de N al cuadrado.
Algoritmo de tiempo exponencial:
Es aquel que en el peor de los casos permite solucionar cierto problema en un tiempo determinado por una función exponencial con el tamaño de la entrada.
Por ejemplo, un algoritmo que para ordenar una lista de N elementos pueda llegar a necesitar 2^N segundos en el peor caso, sería exponencial.
Es sencillo comprobar que un algoritmo de tiempo polinomial siempre termina siendo más rápido que uno de tiempo exponencial si el tamaño de la entrada es lo suficientemente grande.

Valor de N	N^5	2^N
5	3.125	32
10	100.000	1.024
15	759.375	32.768
20	3.200.000	1.048.576
25	9.765.625	33.554.432
30	24.300.000	1.073.741.824
35	52.521.875	34.359.738.368
40	102.400.000	1.099.511.627.776

Puedes ver más información sobre complejidad computacional en el siguienteenlace:
Teoría de la complejidad computacional
Puedes ver más información sobre análisis de algoritmos en el siguiente enlace:
Análisis de algoritmos

2. Analítica y búsqueda de respuestas.

En esta sección tendremos un primer contacto con la analítica de datos, incluyendo sus distintos niveles, y estudiaremos las principales metodologías que se emplean en el campo de la Minería de Datos.
Veremos aquí un esquema/resumen para que puedas tener una vista general:

- **Niveles de Analítica de Datos:**
  - **Análisis Descriptivo:** ¿qué ha ocurrido?
  - **Análisis Diagnótico:** ¿por qué ha ocurrido?
  - **Análisis Predictivo:** ¿qué va a ocurrir?
  - **Análisis Prescriptivo:** ¿qué hacer para que algo ocurra?
- **Principales metodologías en Minería de Datos:**
  - **SEMMA: Sample, Explore, Modify, Model, Assess.**
  - **CRISP-DM:** Cross-industry standard process for data mining.

2.1. Análisis de datos y analítica de datos.

Una vez que somos capaces de adquirir, integrar y gestionar grandes cantidades de datos, el siguiente paso es tratar de obtener algún tipo de valor de tal información.
Llegados a este punto, es importante diferenciar entre lo que es Analítica de Datos y el Análisis de Datos, ya que se trata de conceptos muy íntimamente relacionados pero no idénticos ni intercambiables.
Análisis de Datos:
El análisis de datos consiste en encontrar hechos, relaciones, patrones, revelaciones y/o tendencias, por lo general con la intención de poder apoyar en la mejor medida posible la toma de decisiones.
Por ejemplo, una compañía puede querer analizar las ventas de guantes de lana para determinar cómo están relacionadas con la temperatura ambiente que hace cada día y de ese modo tomar las mejores decisiones posibles a la hora de proveerse de guantes en función de la predicción del tiempo.
Analítica de Datos:
La analítica de datos es un concepto más amplio, ya que incluye dentro al propio análisis de datos así como al resto de actividades alrededor del ciclo de vida del dato (muchas de las cuales ya hemos visto).
Algunas de esas actividades, entre otras, son:

- Obtención/recolección de los datos desde diversas fuentes.
- Limpieza.
- Integración (desde las diversas fuentes a su representación unificada).
- Gobierno de Datos, mediante el cual se gestiona:
  - Disponibilidad (datos disponibles cuando van a usarse).
  - Usabilidad (datos válidos para lo que se quiere hacer con ellos).
  - Integridad (datos correctos).
  - Seguridad (de modo que no puedan ser accedidos ni desde fuera de la organización ni por quienes no tienen los apropiados permisos dentro de la propia organización).
- Análisis de Datos (ya mencionado).

Algunos usos comunes de la Analítica de Datos son los siguientes:

- En el mundo de los negocios, para disminuir los costes operacionales y facilitar la toma estratégica de decisiones.
- En el mundo científico, para entender por qué ocurre un determinado fenómeno y así mejorar el modo de abordarlo.
- En el mundo de los servicios, para disminuir costes y mejorar la calidad de los mismos.

2.2. Niveles de analítica de datos.

Dentro de la Analítica de Datos existen 4 categorías o niveles bien diferenciados:

Análisis Descriptivo:

¿Qué ha ocurrido?
El Análisis Descriptivo intenta describir lo que ha ocurrido.
Por lo general produce como resultado reportes o cuadros de mando estáticos obtenidos mediante consultas a almacenes de datos operacionales (tipo OLTP), como los CRM o ERP empresariales.
Intenta responder a preguntas del tipo de:

- ¿Cuál fue el beneficio mensual de la compañía durante los últimos 12 meses?
- ¿Qué tendencia tiene el número de llamadas de queja que estamos recibiendo en el soporte telefónico a clientes?
- ¿Qué línea de producto está produciendo los mejores resultados?

Análisis Diagnóstico:

¿Por qué ha ocurrido?

El Análisis Diagnóstico intenta determinar la causa de un fenómeno que ha ocurrido o está ocurriendo, detectando qué información resulta estar relacionada con el mismo.
Por lo general implica obtener información de diversas fuentes y almacenarla en estructuras específicas (tipo OLAP) que facilitan su análisis. Los resultados de tal análisis se muestran mediante herramientas interactivas de visualización que permiten a los usuarios identificar tendencias y patrones.
Intenta responder a preguntas del tipo de:

- ¿Por qué tenemos más quejas desde el Norte de España que desde el Sur?
- ¿Por qué estamos vendiendo menos neveras este año?
- ¿Por qué este mes hemos batido el récord de bajas de clientes?

Como puedes imaginar, el Análisis Diagnóstico tiene mayor complejidad que el Análisis Descriptivo, pero gracias a él podemos obtener un mayor valor de los datos.

Análisis Predictivo:

¿Qué ocurrirá?
El Análisis Predictivo intenta predecir qué ocurrirá en un futuro, gracias a la generación de modelos predictivos fruto de procesos de tipo Machine Learning, con la intención de ser capaces de identificar tanto riesgos como oportunidades.
Intenta responder a preguntas del tipo de:

- ¿Qué probabilidad hay de que un potencial cliente responda positivamente a esta oferta?
- ¿Qué probabilidad hay de que este tratamiento cure una determinada enfermedad en determinado rango de edades?
- ¿Si el cliente se ha interesado en este producto, con qué probabilidad le interesará este otro?

Como puedes imaginar, el Análisis Predictivo tiene mayor complejidad que el Análisis Diagnóstico, pero gracias a él podemos obtener un mayor valor de los datos.

Análisis Prescriptivo:

¿Qué hacer para que ocurra?
El Análisis Prescriptivo se apoya en los resultados que es capaz de producir el Análisis Predictivo, probando automáticamente diversas posibles acciones alternativas para así ser capaz de prescribir la mejor acción a tomar. No sólo se enfoca en la acción prescrita sino que también trata de dar información al usuario acerca de la razón por la que es la mejor.
Intenta responder a preguntas del tipo de:

- ¿Cuál de estos productos funcionará mejor si lo vendemos en Canadá?
- ¿Cuál es el mejor mes para lanzar nuestro nuevo servicio?

Como puedes imaginar, el Análisis Prescriptivo tiene mayor complejidad que el Análisis Predictivo, pero gracias a él podemos obtener un mayor valor de los datos.

2.3. Principales metodologías de minería de datos.

En esta sección conoceremos las metodologías más empleadas en el campo de la Minería de Datos.
Como ya sabrás por los contenidos de anteriores unidades de trabajo, la Minería de Datos es una rama de la Inteligencia Artificial que busca obtener valor del dato fundamentalmente mediante la generación de modelos predictivos. Por lo tanto se emplea para realizar Análisis Predictivo, llegando sus resultados a emplearse también posteriormente si realizamos Análisis Prescriptivo.
Veremos aquí un esquema/resumen para que puedas tener una vista general:

SEMMA (Sample, Explore, Modify, Model, Assess):
Una lista de pasos secuenciales desarrollada por SAS Institute que se emplean en muchas ocasiones a modo de metodología para la minería de datos.
CRISP-DM (Cross-industry standard process for data mining):
Una metodología ampliamente empleada para el proceso de minería de datos que goza de una cierta oficialidad.

2.3.1. SEMMA.

***SEMMA* es el acrónimo empleado para una lista de pasos secuenciales desarrollada por *SAS Institute*, uno de los principales fabricantes de software para inteligencia empresarial.**
Los pasos de *SEMMA* y sus tareas relacionadas son:

- **(S)ample:** Seleccionar los datos para el modelado mediante muestreo (*data sampling*). El conjuntode datos debe quedar con tamaño como para contener suficiente información, pero tan pequeño como para poder usarse de modo eficiente.
- **(E)xplore:** Ayudarse de la visualización de datos para entender los datos y encontrar relaciones entre las variables así como anormalidades.
- **(M)odify:** Seleccionar, crear y transformar variables como preparación para el modelado de datos.
- **(M)odel:** Aplicación de diversas técnicas de minería de datos para producir modelos.
- **(A)ssess:** Evaluación de los modelos para comprobar su utilidad práctica.

Los 5 pasos de *SEMMA* se emplean en muchas ocasiones a modo de metodología para la minería de datos. Sin embargo, voces críticas indican que para la fase *Sample* es necesario un conocimiento profundo no sólo de los datos sino de los aspectos de negocio aplicables, lo cual no aparece contemplado como una de las tareas dentro esa fase.
En todo caso, sus creadores en *SAS Institute* avisan de que realmente no idearon los pasos como una metodología para minería de datos, sino que simplemente constituyen la organización lógica del conjunto de herramientas funcionales que emplea uno de sus productos para minería de datos, llamado *SAS Enterprise Miner*.

Puedes ver más información sobre *SEMMA* en el siguiente enlace:

SEMMA

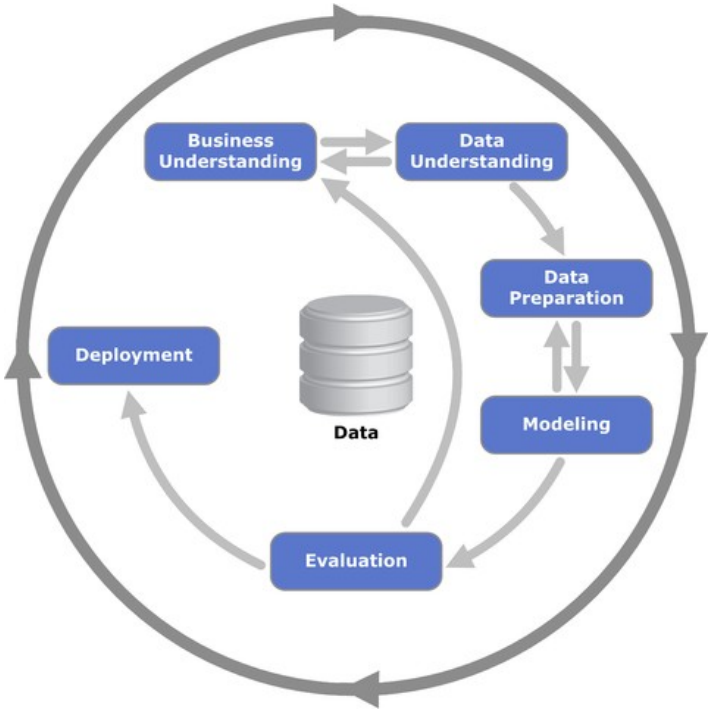
2.3.2. CRISP-DM.

CRISP-DM (o *Cross-industry standard process for data mining*) es una metodología ampliamente empleada para el proceso de minería de datos que goza de una cierta oficialidad al provenir de un proyecto de la Unión Europea bajo la iniciativa ESPRIT.
El *CRISP-DM* divide el proceso de minería de datos en 6 fases:

- Comprensión del negocio.
- Comprensión de los datos.
- Preparación de los datos.
- Modelado.
- Evaluación.
- Despliegue.

La clave de la metodología *CRISP-DM* es que no se trata de una secuencia sino de un proceso cíclico, ya que propone continuar dando vueltas por las fases (aplicando las lecciones aprendidas durante cada iteración en el ciclo para la siguiente).
Además de su naturaleza cíclica existen diversas transiciones posibles entre las fases que contempla, como se puede ver en el diagrama que resume la metodología.

Diagrama del proceso CRISP-DM



Puedes ver más información sobre *CRISP-DM* en el siguiente enlace:

Cross Industry Standard Process for Data Mining

3. R para analizar datos.

R es un lenguaje de programación interpretado de código abierto creado específicamente para facilitar el análisis de datos. Su nombre proviene de que está basado en un lenguaje preexistente llamado S.

Nota: El software gracias al cual podemos ejecutar programas escritos en tal lenguajetambién se llama R.

Algunas de sus peculiaridades son las siguientes:

- Es **interpretado**: no es necesario compilar los programas sino que son interpretados(por otro programa llamado intérprete).
- Es de **alto nivel**: no se tiene acceso a bajo nivel a la máquina en la que se ejecuta(gestión de la memoria, etc), sino que todo está encapsulado para que el programador pueda concentrarse en el análisis de datos.
- Es un lenguaje **imperativo** (con ejecución de código línea a línea) pero soporta programación orientada a objetos (para analizar datos usaremos clases constantemente) y en cierto modo también permite programación funcional (podemos crear funciones, llamarlas de forma recursiva y usarlas como si fuesen variables).

R puede obtenerse en su web oficial (https://www.r-project.org/) de modo que podemos trabajar con R de forma totalmente interactiva mediante su intérprete o podemos ejecutar programas que hayamos escrito con cualquier editor de texto plano.

Además, podemos instalar uno de los muchos IDE que están disponibles, siendo RStudio uno de los más utilizados. RStudio puede descargarse desde su web oficial https://www.rstudio.com/.

Si el alumno no desea o no puede instalar R en su ordenador, puede emplear uno de los varios intérpretes de R disponibles online. Te recomendamos los 2 siguientes, ya que permiten crear gráficos para visualización.

myCompiler - Online R Compiler
rdrr.io

Puedes ver más información sobre el lenguaje de programación R en la propia página oficial de R:

Página oficial de R

También puedes consultar un resumen sobre R en el siguiente enlace:

R (lenguaje de programación)

3.1. Primeros pasos.

En este curso no vamos a pretender que el alumno se convierta en un experto en R, sino que conozca algunas bases que le permitan entender qué características tiene un lenguaje especialmente creado para análisis de datos.
Por ello, vamos a limitarnos a dar unos primeros pasos muy guiados que puedes ir ejecutando en tu intérprete de R (ya sea uno instalado o un intérprete en línea entre los que te hemos recomendado).
Puedes probar a hacer ejecutar código con tus propias modificaciones para entender mejor qué es lo que está ocurriendo y cómo funciona el lenguaje.

```
Sum(1:3)
```

```
[1] 6
```

Para entender este ejemplo en primer lugar debemos saber que R puede trabajar tanto con valores individuales como con **vectores** (que son conjuntos ordenados de valores).
El operador ":" crea una secuencia de números (en este caso entre 1 y 3), con lo cual tenemos un vector.
La función *sum* suma los elementos del vector que recibe como parámetro, y por ello el resultado es 6.

Obtener ayuda:

R integra un mecanismo para solicitar ayuda acerca de funciones y construcciones del lenguaje.

```
?sum                    #abre la página de ayuda de la función
sum?"+"          #abre la página de ayuda de la operación de suma
?"if"            #abre la página de ayuda acerca de la construcción del lenguaje
"if"??plotting          #busca en la ayuda temas que contengan la palabra "plotting"
```

Asignación de variables:

```
x <- 3
y = 4
x+y
[1] 7
```

Podemos asignar valores a variables usando = o <-, aunque por razones históricas se prefiere ←.

Mostrar resultados:

```
x <- 3 + 4
```

Cuando hacemos una asignación por defecto no vemos el valor que recibe la variable.

```
(x <- 3 + 4)

[1] 7
```

Sin embargo, podemos englobar en paréntesis para solicitarle a R que nos muestre el valor asignado.

Operaciones con vectores:

```
1:3 + 4:6
[1] 5 7 9
```

Aplicamos la operación de suma sobre dos vectores, y el resultado es otro vector cuyas posiciones se obtienen de sumar los valores de los vectores de entrada en esa misma posición.

```
C(1,3,5)
[1] 1 3 5
```

La función especial *c* crea un vector con los parámetros recibidos.

```
C(2,3,4) -1
[1] 1 2 3
```

Si realizamos una operación con un vector y un número, el resultado es el vector resultante de aplicar la operación en cada una de las posiciones.

```
C(2, 5, 4-2, 7) == 3
[1] FALSE FALSE FALSE FALSE
```

Usamos el operador == para comprobar igualdad (< para "menor que", > para "mayor que",<= para "menor o igual que", >= para "mayor o igual que").
La operación se realiza sobre todos los elementos del vector.

```
x <- c(1,2,3)
y = c(5,6,7)
x*y
[1] 5 12 21
```

Por supuesto, también podemos asignar vectores a variables.

```
Length(1:8)
[1] 8
```

Podemos obtener la longitud de un vector.

```
a <- c("blanco","negro","azul")
length(a)
```

```
nchar(a)
[1] 3
[1] 6 5 4
```

Si el vector contiene cadenas de caracteres, su longitud es el número de cadenas. Si queremos conocer la longitud de cada cadena usaremos la función *nchar*.

```
a <- c(manzana=4,naranja=3,mandarina=5)
a

manzana     naranja     mandarina
4           3           5

x <- c(1,2,3)
names(x) <- c("uno","dos","tres")
x

uno   dos   tres
1     2     3
```

Podemos asignar nombres a los elementos de un vector.

```
X <- 2:7
x
x[c(1,3,5)]

[1] 2 3 4 5 6 7
[1] 2 4 6
```

Podemos acceder a un vector mediante sus posiciones (índices).
Ten en cuenta que R el índice del primer elemento es el 1 (en contraposición a muchos lenguajes en los que es el 0).

```
x <- 2:7
x
x[c(-1,-3,-5)]

[1] 2 3 4 5 6 7
[1] 3 5 7
```

Podemos acceder a un vector excluyendo ciertas posiciones.

```
X <- 1:6
x[c(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE)]

[1] 1 3 4 6
```

Podemos acceder a un vector mediante un vector de valores booleanos que nos indica con TRUE en las posiciones que queremos y FALSE en las que no.

Números especiales:

```
c(-Inf+1,Inf-1,Inf-Inf,NA*5)

[1] -Inf Inf NaN NA
```

R soporta los siguientes números especiales:

Inf: infinito.
-Inf: menos infinito.
NaN: *not-a-number*, indicando que una operación no tiene sentido o no puedo ejecutarse correctamente.
NA: *not available*, representando un valor desconocido (lo cual es muy común en análisis de datos).

3.2. Arrays y matrices.

Hasta ahora hemos visto que R trabaja sin problemas con vectores, que son objetos unidimensionales.
Sin embargo, también podemos trabajar con conjuntos de datos multidimensionales mediante arrays. Hay que tener en cuenta que los arrays son rectangulares (es decir que todas las filas deben tener la misma longitud, al igual que todas las columnas y todas las demás posibles dimensiones).

```
(array_3d <- array(
1:24,
dim = c(2, 3, 4),
dimnames = list(
c("izquierda", "derecha"),
c("primero", "segundo", "tercero"),
c("uno", "dos", "tres", "cuatro")
)
))

, , uno

          primero segundo tercero
izquierda       1       3       5
derecha         2       4       6

, , dos

          primero segundo tercero
izquierda       7       9      11
derecha         8      10      12

, , tres

          primero segundo tercero
izquierda      13      15      17
derecha        14      16      18

, , cuatro
```

```
                primero segundo tercero
izquierda       19      21      23
derecha         20      22      24
```

Podemos crear un array de cualquier número de dimensiones mediante la función *array*,indicándole un vector con todos los valores, otro con la longitud en cada dimensión (lo cual implica cuántas dimensiones queremos) y una lista de vectores con los nombres para las distintas dimensiones.

```
dim(array_3d)
length(dim(array_3d))
```

```
[1] 2 3 4
[1] 3
```

Si tenemos un array podemos saber cuántas dimensiones tiene y la longitud de cada una con la función *dim*.

```
array_3d[,1:2,c(2,3)]
, , dos
```

```
                primero segundo
izquierda       7       9
derecha         8       10
```

```
, , tres
```

```
                primero segundo
izquierda       13      15
derecha         14      16
```

Podemos acceder al array utilizando corchetes "[]" de un modo similar a como lo hacemos con los vectores, pero usando comas para separar las distintas dimensiones (si en una posición no ponemos nada -vacío- obtendremos su contenido completo).

Las **matrices** son un caso especial de arrays limitado a 2 dimensiones.

```
(matriz <- matrix(
1:6,
nrow = 3,                       #ncol = 2 hace lo mismo
dimnames = list(
c("uno", "dos", "tres"),
c("arriba", "abajo")
)
))
```

```
        arriba abajo
uno     1       4
dos     2       5
tres    3       6
```

Podemos crear una matriz con la función matrix, pasándole como argumentos un vector con los números, el número de filas (o de columnas) y una lista de vectores con los nombres para las distintas dimensiones.

```
matriz[,2]
```

	uno	dos	tres
	4	5	6

El acceso a las posiciones de la matriz es igual que en el caso de los arrays, teniendo en cuenta que sólo tienen 2 dimensiones.

```
matriz2 <- matrix(
3:8,
nrow = 3,                          #ncol = 2 hace lo mismo
dimnames = list(
c("uno", "dos", "tres"),
c("arriba", "abajo")
)
)

matriz + matriz2
```

	arriba	abajo
uno	4	10
dos	6	12
tres	8	14

Podemos realizar operaciones de álgebra de matrices.

3.3. Listas y dataframes.

Vectores, arrays y matrices tienen el inconveniente de que sólo permiten contener elementos de un único típo (por ejemplo todos numéricos o todos cadenas de caracteres).

Una lista es algo equivalente a un vector en el que cada elemento puede ser de un tipo distinto.

```
(lista <- list(
c(1, 2, 3),
33,
matrix(c(6, 7, 14, 1), nrow = 2),
mean
))

[[1]]
[1] 1 2 3

[[2]]
[1] 33

[[3]]
     [,1] [,2]
[1,]    6   14
[2,]    7    1
```

```
[[4]]
function (x, ...)
UseMethod("mean")
```

Podemos crear una lista mediante la función *list*, pasándo le como argumentos los distintos valores que queramos incluír en ella.
En este ejemplo hemos creado la lista con un vector, un valor numérico, una matriz y una función (recordemos que R permite usar funciones como si fuesen variables).

```
names(lista) <- c("vector", "numero", "matriz", "funcion mean")
lista

$vector
[1] 1 2 3

$numero
[1] 33

$matriz
     [,1] [,2]
[1,]    6   14
[2,]    7    1

$`funcion mean`
function (x, ...)
UseMethod("mean")
```

De igual modo que con los vectores, podemos asignar nombres a sus elementos.

```
Lista[2:3]

$numero
[1] 33

$matriz
     [,1] [,2]
[1,]    6   14
[2,]    7    1
```

Podemos acceder a la lista igual que hacíamos con los vectores.

Los ***dataframes*** son el equivalente a matrices (por lo tanto bidimensionales) que pueden almacenar distintos tipos de datos, siendo por lo tanto similares al contenido de una hoja de cálculo.
Por ello están especialmente indicados para trabajar con conjuntos de datos en su formato más común: secuencias de filas, las cuales están formadas por columnas que pueden contener distintos tipos de datos.

```
V <- 1:5
(mi_dataframe <- data.frame(
x = letters[1:5],
```

```
y = v,
z = v > 2
))

x        y       z
1        a       1 FALSE
2        b       2 FALSE
3        c       3 TRUE
4        d       4 TRUE
5        e       5 TRUE
```

Podemos crear dataframes mediante la función *data.frame*, pasándo le los datos por columnas (todas de la misma longitud). Como podemos ver, cada columna puede contener distinto tipo de datos (pero toda la columna el mismo).

```
mi_dataframe[2:3,c(1,2)]
x y
2 b 2
3 c 3
```

Podemos acceder a los dataframes de modo análogo a como lo hacemos con matrices.

```
mi_dataframe2 <- data.frame(
col4 = 6:10,
col5 = v+3,
col6 = v < 3
)

cbind(mi_dataframe, mi_dataframe2)
x y z col4 col5 col6
1 a 1 FALSE 6 4 TRUE
2 b 2 FALSE 7 5 TRUE
3 c 3 TRUE 8 6 FALSE
4 d 4 TRUE 9 7 FALSE
5 e 5 TRUE 10 8 FALSE
```

Podemos realizar distintas operaciones con dataframes, como por ejemplo enlazarlos por columnas con *cbind*.

```
mi_dataframe3 <- data.frame(
x = letters[6:8],
y = 6:8,
z = c(TRUE,FALSE,TRUE)
)

rbind(mi_dataframe, mi_dataframe3)
x y z
1 a 1 FALSE
2 b 2 FALSE
3 c 3 TRUE
4 d 4 TRUE
5 e 5 TRUE
```

6 f 6 TRUE
7 g 7 FALSE
8 h 8 TRUE

De modo similar, también podemos enlazar dos dataframes por filas con *rbind*.

3.4. Análisis y visualización.

Respecto del análisis de datos, en primer lugar hemos de tener en cuenta que R tiene una gran cantidad de conjuntos de datos dentro de paquetes a los cuales podemos acceder si están instalados.
Para acceder a ellos, podemos emplear la función *data* pasándo le como argumentos el nombre del dataset y el paquete en el que éste se encuentra.

```
data("kidney", package = "survival")
head(kidney)
```

```
id time status age sex disease frail
1 1 8 1 28 1 Other 2.3
2 1 16 1 28 1 Other 2.3
3 2 23 1 48 2 GN 1.9
4 2 13 0 48 2 GN 1.9
5 3 22 1 32 1 Other 1.2
6 3 28 1 32 1 Other 1.2
```

En este caso hemos cargado el dataset llamado *kidney* dentro del paquete *survival* y hemos mostrado las primeras filas mediante la función *head*.

```
dim(kidney)
```

```
[1] 76 7
```

Vemos que el dataset contiene 76 filas y 7 columnas.

```
time <- kidney$time
mean(time)
median(time)
var(time)
sd(time)
```

```
[1] 101.6316
[1] 39.5
[1] 17138.5
[1] 130.9141
```

R nos permite calcular diversas estadísticas utilizando funciones específicas para ello.
Fíjate que para este ejemplo hemos obtenido una columna del dataframe utilizando el accesor $ tras su nombre seguido del nombre de la columna.

```
summary(kidney)
```

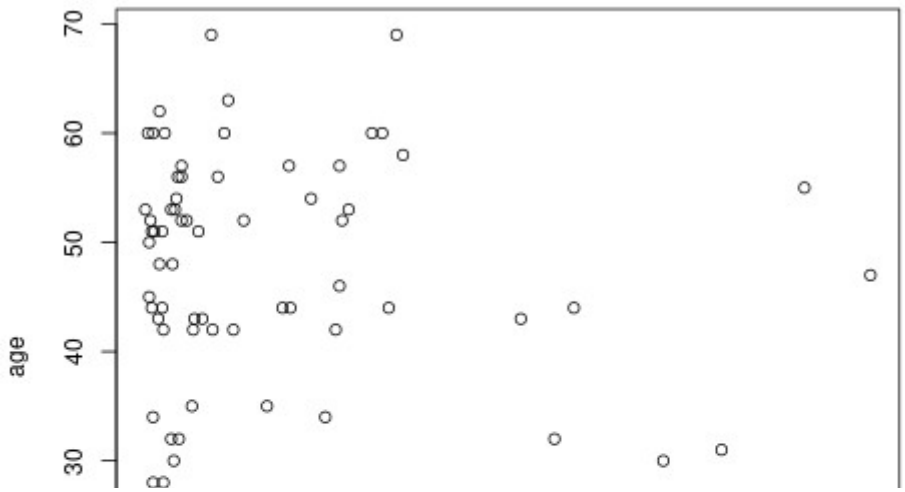
```
id                time            status           age              sex
```

Min. :	1.0	Min. : 2.0	Min. : 0.0000	Min. : 10.0	Min. : 1.000
1st Qu.:	10.0	1st Qu.: 16.0	1st Qu.:1.0000	1st Qu.:34.0	1st Qu.:1.000
Median :	19.5	Median : 39.5	Median :1.0000	Median :45.5	Median :2.000
Mean :	19.5	Mean : 101.6	Mean : 0.7632	Mean : 43.7	Mean :1.737
3rd Qu.:	29.0	3rd Qu.:149.8	3rd Qu.:1.0000	3rd Qu.:54.0	3rd Qu.:2.000
Max. :	38.0	Max. : 562.0	Max. : 1.0000	Max. :69.0	Max. :2.000

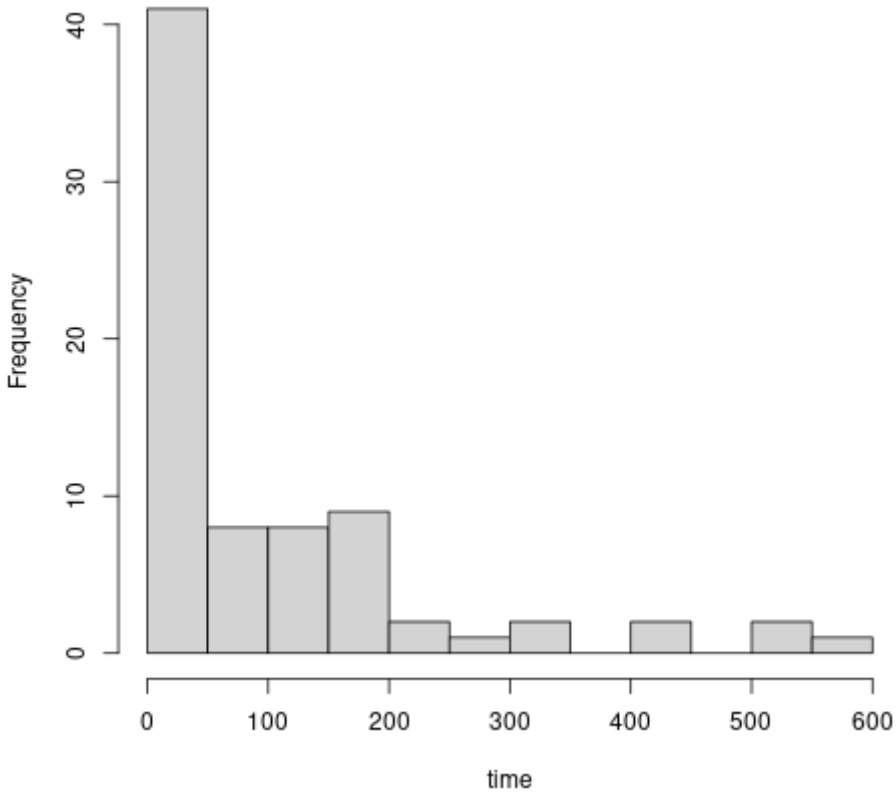
disease		frail
Other:	26	Min. :0.200
GN :	18	1st Qu.:0.600
AN :	24	Median :1.100
PKD :	8	Mean :1.184
3rd Qu.:	1.500	Max. :3.000

Incluso podemos utlizar la función *summary* para obtener una serie de estadísticas interesantes acerca de todas las columnas del dataframe.

with(kidney, plot(time, age))



with(kidney, hist(time))



Como podemos ver, R nos permite visualizar datos de un modo muy sencillo. Aquí mostramos gráficos de tipo *plot* (diagrama de puntos) e *hist* (histograma), pero existenmuchos otros.
También hay que tener en cuenta que aunque aquí hemos usado el sistema básico de gráficos de R, hay otras dos librerías más avanzadas, llamadas *lattice* y *ggplot2*, las cuales permiten crear visualizaciones más avanzadas, detalladas y coloridas.

Para saber más acerca de análisis exploratorio con R, te recomendamos que visites el siguiente *notebook* en Kaggle (una web muy interesante para científicos de datos).

Exploratory Data Analysis - R

4. Python para analizar datos.

4.1. Primeros pasos.
4.2. Arrays con NumPy.
4.3. Análisis con Pandas.
4.4. Visualización con Matplotlib.
4.5.
4.6.